[illegible]

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/079,928	02/19/2002	Martin D. Richek	3434-P02437US1	6502

110 7590 02/27/2007
DANN, DORFMAN, HERRELL & SKILLMAN
1601 MARKET STREET
SUITE 2400
PHILADELPHIA, PA 19103-2307

EXAMINER	
RAMPURIA, SATISH	

ART UNIT	PAPER NUMBER
2191	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
2 MONTHS	02/27/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

MAILED

FEB 26 2007

Technology Center 2100

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 10/079,928
Filing Date: February 19, 2002
Appellant(s): RICHEK, MARTIN D.

Niels Haun

For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed October 26, 2006 appealing from the Office action (Final Rejection) mailed April 21, 2006.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

The following are the related appeals, interferences, and judicial proceedings known to the examiner which may be related to, directly affect or be directly affected by or have bearing on the board's decision in the pending appeal:

Based on the information supplied by the Appellants, and to the best of Appellants' legal representative's knowledge, the real party in the interest is the assignee, Quazant Technology, Inc. of Cypress, TX.

(2) Related Appeals and Interferences

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The amendment after final rejection filed on August 11, 2006.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

6,144,965	Oliver	11-2000

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless --
(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 3-10, 12-13, 15-23, 25-28, 30-35 are rejected under 35 U.S.C. 102(e) as being anticipated by US Patent No. 6,144,965 to Oliver (hereinafter called Oliver).

Per claim 3:

Oliver disclose:

- A computer-implemented method of memory management, comprising the steps of:
- providing a smart pointer (A smart pointer element is analogous to LISTPTR, See FIG. 5C, element LISTPTR)
- for association with a memory-resident element (FIG. 5C, element OBJECT) (col. 5, lines 12-13 "second entry in the pointer list is created for the same object" / associating an additional smart pointer element),
- the smart pointer including a next pointer (FIG. 5C, element NEXT and col. 5, lines 15-16 ""the second pointer… includes a "next pointer"");

- providing an assignment means for assigning the next pointer (FIG. 5C, element NEXT – 'next pointer' connects the elements of the circular linked list)
- to point to the smart pointer thereby creating a linked list comprising the smart pointer; (col. 5, lines 17-20 "the "next pointer"... linked to each other" / using a next pointer, pointer elements are linked in a circular linked list / see Fig. 5C) (A circular linked list of smart pointer [elements] associated with (pointing to / referencing) a 'memory resident element' object. Each smart pointer [element] in the list has a 'next pointer', pointing to the next smart pointer [element].)

- providing a comparison means for comparing the value of the next pointer to the value of the memory location of the smart pointer in which the next pointer is included, (col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer"...are examined..." / compare the value of the next pointer, the address the next pointer is pointing to, to the value of the location of the element in the linked list / the address of the smart pointer element in the linked list)
- whereby a determination can be made if the linked list contains more than one smart pointer (col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer"...are examined (compared)... If the "next pointer" pointer is the same as the "previous pointer" (if the 'next pointer' is pointing to an address containing an pointer element in a circular linked list of one pointer element / pointing to itself) "...then there is clearly only one pointer (element) remaining in the list... final pointer is deleted, no pointers will remain in the list. Thus, if the "next pointer" pointer is the same as the "previous pointer" pointer, then the object... deleted...

and the last pointer to the object... deleted... If the "next pointer" pointer is not the same as the "previous pointer" (in the case where there are more than one pointer elements in the linked list)... other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506"); and

- deleting the memory-resident element associated with the smart pointer (col. 5, lines 28-29 "the object (memory resident element) is then deemed unreferenced and may be deleted")
- if the value of the next pointer of the smart pointer is equal to the value of the memory location of the smart pointer in which the next pointer is included (If 'next pointer' is wrapped around circular linked list of one element and is pointing to itself / both have the same address value, as in the case of only one pointer element left in the circular linked list of pointer elements, col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer" and "previous pointer" pointers are examined (compared)... If the "next pointer" pointer is the same as the "previous pointer" then there is clearly only one pointer remaining in the list... final pointer is deleted...") and
- not deleting the memory-resident element if the value of the next pointer of the smart pointer is not equal to the value of the memory location of the smart pointer in which the pointer is included (in the case of more than one pointer element remaining in the circular linked list, 'next pointer' is pointing to an address value of a successive pointer element in the circular linked list, addresses are not the

same - col. 5, lines 43-46 “If the "next pointer" pointer is not the same as the "previous pointer" pointer, however, then there are clearly other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506”).

(When the ‘next pointer’ (of the smart pointer [element]) is pointing to itself, as determined by a compare of the memory location values, in the case of a circular linked list of only one element, then delete the ‘memory resident element’ (object pointed to by smart pointer elements)

Otherwise do not delete the ‘memory resident element’, if the values of the memory location are different / meaning there is more than one smart pointer [element] left in the circular linked list that is referencing the ‘memory resident element’.

This is a way of accounting for multiple references to a memory location holding an object and knowing when the use of the memory space occupied by the object is no longer required.

As references to an object are no longer needed, a pointer element may be deleted (col. 5, line 22) from the circular linked list. The ‘next pointer’ is adjusted to keep the circular characteristic of the linked list.

When the 'next pointer' points to the (col. 5, line 25) one and only element left in the circular linked list of pointer elements, the 'values of the memory location' are the same.

If it is desired to delete (memory management) the last linked list pointer (when the next pointer points to the one and only linked list element / points to itself) to the 'memory resident element'/object, the 'memory resident element' / object referenced by the pointer linked list may be deleted, as it is no longer needed.)

Per claim 4:

The rejection of claim 3 is incorporated, and further, Oliver disclose:

- wherein the method comprises the step of providing a common base to the smart pointer (col. 2, lines 7-8 "base class creates a reference counter for smart pointer to the object").

Per claim 5:

The rejection of claim 3 is incorporated, and further, Oliver disclose:

- wherein the element is an object in an object-oriented programming environment (col. 3, lines 28-30 "a method... in an object-oriented programming environment").

Per claims 6:

The rejection of claim 5 is incorporated, and further, Oliver disclose:

- wherein the smart pointer includes an object pointer for pointing to the object (col. 4, lines 42-44 "a copy of the original reference pointer is mad, the new reference pointer also points to the original object and its associated count object").

Per claims 7, 10, 17, 20, 25, and 28:

The rejection of claim 3 is incorporated, and further, Oliver disclose:

- wherein the linked list comprises a ring (see Fig. 5C and related discussion).

Per claim 8:

Oliver disclose:

- wherein the smart pointer includes a previous pointer. The limitations in the claims are similar to those in claim 3, and rejected under the same rational set forth in connection with the rejection of claim 3.

Per claims 9, 19, and 27:

The rejection of claim 8 is incorporated, and further, Oliver disclose:

- providing an assignment means for assigning the previous pointer to point to the smart pointer, thereby creating a bi-directional, doubly-linked list (see Fig. 5B and related discussion).

Per claim 21:

The rejection of claim 3 is incorporated, and further, Oliver disclose:

- providing a deletion means for deleting the memory-resident element associated with the smart pointer (col. 5, lines 22-23 "delete a pointer... is deleted") if the value of the next-pointer of the smart pointer is equal to the value of the memory location of the smart pointer in which the next-pointer is included (col. 4, lines 55-67 "each time a pointer is deleted... deleted in step 408").

Per claims 12, 22, and 30:

The rejection of claim 3 is incorporated, and further, Oliver disclose:

- wherein the smart pointer includes a first smart pointer, and wherein the method comprises the step of providing an attachment means for attaching a second smart pointer associated with the memory-resident element to the linked list element (col. 5, lines 12-20 "second entry in the pointer list is created for the same object... linked to each other").

Per claim 13:

Oliver disclose:

- providing a linked list comprising a smart pointer (FIG. 5C, element NEXT – 'next pointer' connects the elements of the circular linked list) associated with a memory-resident element (col. 5, lines 12-13 "second entry in the pointer list is created for the same object"),
- the smart pointer including a next-pointer for pointing to the smart pointer (col. 5, lines 17-20 "the "next pointer"… linked to each other" / using a next pointer, pointer elements are linked in a circular linked list / see Fig. 5C)
  (A circular linked list of smart pointer [elements] associated with (pointing to / referencing) a 'memory resident element' object. Each smart pointer [element] in the list has a 'next pointer', pointing to the next smart pointer [element].); and
- providing a comparison means for comparing the value of memory of the smart pointer to the value of the next pointer of the smart pointer, to provide whether the linked list contains only the smart pointer (col. 5, lines 34-37 "pointer… examined… pointer is the same…one pointer remaining in the list" / compare the value of the next pointer, the address the next pointer is pointing to, to the value of the location of the element in the linked list / the address of the smart pointer element in the linked list)
- deleting the memory-resident element associated with the smart pointer (col. 5, lines 28-29 "the object (memory resident element) is then deemed unreferenced and may be deleted" and col. 5, lines 22-23 "delete a pointer… is deleted") if the value of the next pointer of the smart pointer is equal to the value of the memory

location of the smart pointer in which the next pointer is included (If ‘next pointer’ is wrapped around circular linked list of one element and is pointing to itself / both have the same address value, as in the case of only one pointer element left in the circular linked list of pointer elements, col. 5, lines 34-37 “FIG. 5D… Prior to deleting a pointer, the "next pointer" and "previous pointer" pointers are examined (compared)… If the "next pointer" pointer is the same as the "previous pointer" then there is clearly only one pointer remaining in the list… final pointer is deleted) and

- not deleting the memory-resident element if the value of the next pointer of the smart pointer is not equal to the value of the memory location of the smart pointer in which the pointer is included (col. 5, lines 43-46 “If the "next pointer" pointer is not the same as the "previous pointer" pointer, however, then there are clearly other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506”).

(When the ‘next pointer’ (of the smart pointer [element]) is pointing to itself, as determined by a compare of the memory location values, in the case of a circular linked list of only one element, then delete the ‘memory resident element’ (object pointed to by smart pointer elements)

Otherwise do not delete the ‘memory resident element’, if the values of the memory location are different / meaning there is more than one smart pointer [element] left in the circular linked list that is referencing the 'memory resident element'.

This is a way of accounting for multiple references to a memory location holding an object and knowing when the use of the memory space occupied by the object is no longer required.

As references to an object are no longer needed, a pointer element may be deleted (col. 5, line 22) from the circular linked list. The 'next pointer' is adjusted to keep the circular characteristic of the linked list.

When the 'next pointer' points to the (col. 5, line 25) one and only element left in the circular linked list of pointer elements, the 'values of the memory location' are the same.

If it is desired to delete (memory management) the last linked list pointer (when the next pointer points to the one and only linked list element / points to itself) to the 'memory resident element'/object, the 'memory resident element' / object referenced by the pointer linked list may be deleted, as it is no longer needed.)

Per claim 15:

The rejection of claim 13 is incorporated, and further, Oliver disclose:

- wherein the element is an object in an object-oriented programming environment. The limitations in the claims are similar to those in claim 23, and rejected under the same rational set forth in connection with the rejection of claim 23.

Per claim 16:

- wherein the smart pointer includes an object pointer for pointing to the object. The limitations in the claims are similar to those in claim 13, and rejected under the same rational set forth in connection with the rejection of claim 13.

Per claim 18:

- wherein the smart pointer includes a previous pointer. The limitations in the claims are similar to those in claim 13, and rejected under the same rational set forth in connection with the rejection of claim 13.

Per claims 23 and 35:

- providing a linked list comprising a first smart pointer (A smart pointer element is analogous to LISTPTR, See FIG. 5C, element LISTPTR) and a second smart pointer each associated with a memory-resident element (FIG. 5C, element OBJECT and col. 5, lines 12-13 "second entry in the pointer list is created for the same object" / associating an additional smart pointer element)), the first smart pointer including a first next-pointer (FIG. 5C, element NEXT) for pointing to the second smart pointer and the second smart pointer including a second next-pointer for pointing to the first smart pointer (FIG. 5C, element NEXT – 'next pointer' connects the elements of the circular linked list and col. 5, lines 15-16 "the second pointer... includes a "next pointer""); and

- providing a comparison means for comparing the value of the memory location of a selected smart pointer giving up its association with the memory-resident element to the value of the next-pointer of the selected smart pointer (col. 5, lines 17-20 "the "next pointer"... linked to each other" / using a next pointer, pointer elements are linked in a circular linked list / see Fig. 5C)

  (A circular linked list of smart pointer [elements] associated with (pointing to / referencing) a 'memory resident element' object. Each smart pointer [element] in the list has a 'next pointer', pointing to the next smart pointer [element].),
- to provide a determination whether the linked list contains only the selected smart pointer (col. 5, lines 34-37 "pointer... examined... pointer is the same...one pointer remaining in the list")
- deleting the memory-resident element associated with the smart pointer (col. 5, lines 28-29 "the object (memory resident element) is then deemed unreferenced and may be deleted" and col. 5, lines 22-23 "delete a pointer... is deleted")
- if the value of the next pointer of the smart pointer is equal to the value of the memory location of the smart pointer in which the next pointer is included (col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer" and "previous pointer" pointers are examined (compared)... If the "next pointer" pointer is the

same as the "previous pointer" then there is clearly only one pointer remaining in the list... final pointer is deleted) and

- not deleting the memory-resident element if the value of the next pointer of the smart pointer is not equal to the value of the memory location of the smart pointer in which the pointer is included (col. 5, lines 43-46 "If the "next pointer" pointer is not the same as the "previous pointer" pointer, however, then there are clearly other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506").

(When the 'next pointer' (of the smart pointer [element]) is pointing to itself, as determined by a compare of the memory location values, in the case of a circular linked list of only one element, then delete the 'memory resident element' (object pointed to by smart pointer elements)

Otherwise do not delete the 'memory resident element', if the values of the memory location are different / meaning there is more than one smart pointer [element] left in the circular linked list that is referencing the 'memory resident element'.

This is a way of accounting for multiple references to a memory location holding an object and knowing when the use of the memory space occupied by the object is no longer required.

As references to an object are no longer needed, a pointer element may be deleted (col. 5, line 22) from the circular linked list. The 'next pointer' is adjusted to keep the circular characteristic of the linked list.

When the 'next pointer' points to the (col. 5, line 25) one and only element left in the circular linked list of pointer elements, the 'values of the memory location' are the same.

If it is desired to delete (memory management) the last linked list pointer (when the next pointer points to the one and only linked list element / points to itself) to the 'memory resident element'/object, the 'memory resident element' / object referenced by the pointer linked list may be deleted, as it is no longer needed.)

Per claim 26:

- wherein the first smart pointer and the second smart pointer each include a previous pointer. The limitations in the claims are similar to those in claim 23, and rejected under the same rational set forth in connection with the rejection of claim 23.

Per claim 31:

- comprising the step of providing a common base to the smart pointers. The limitations in the claims are similar to those in claim 23, and rejected under the same rational set forth in connection with the rejection of claim 23.

Per claim 32:

The rejection of claim 23 is incorporated, and further, Oliver disclose:

- wherein the element is an object in an object-oriented programming environment. The limitations in the claims are similar to those in claim 23, and rejected under the same rational set forth in connection with the rejection of claim 23.

Per claim 33:

The rejection of claim 32 is incorporated, and further, Oliver disclose:

- wherein the first smart pointer and the second smart pointer each include an object pointer for pointing to the object. The limitations in the claims are similar to those in claim 23, and rejected under the same rational set forth in connection with the rejection of claim 23.

Per claim 34:

The rejection of claim 32 is incorporated, and further, Oliver disclose:

- wherein the first smart pointer is associated with a first object of a first class and the second smart pointer is associated with a second object of a second class, and wherein the method comprises the step of providing a conversion means for providing automatic conversion between the first smart pointer and the second smart pointer (col. 5, lines 12-22 "second entry... linked to each other").

(10) Response to Argument

In section 7. ARGUMENT (page 13-23), Appellant argued that:

<u>Respecting Claims 3-10, 12</u> (pages 13-14)

Thus, it is also clear from this text that the Examiner (perhaps unintentionally) acknowledges that Oliver compares the next and previous pointers. The "value of the memory location of the smart pointer in which the selected next pointer is included" is not even considered or mentioned in the cited text of Oliver. It is not used in the Oliver test. Only the next and previous pointers are compared in Oliver. Hence, Oliver unquestionably fails to disclose at least Applicant's claimed feature of "comparing the value of the next pointer to *<u>the value of the memory location of the smart pointer</u>* in which the selected next pointer is included." Likewise, Oliver fails disclose the step of "deleting the memory-resident element associated with the smart pointer if the value of the next pointer of the smart pointer is equal to the value of the memory location of the smart pointer in which the next pointer is included and not deleting the memory-resident element if the value of the next pointer of the smart pointer is not equal to the value of the memory location of the smart pointer in which the next pointer is included" as recited in claim 3. For at least these reasons, the rejection of claim 3 in view of Oliver is deficient, as Oliver fails to disclose each and every element recited in claim 3.

<u>Examiner's Response</u>

To find out more about linked lists and pointers please see the attachment (Mastering algorithm with C by Kyle Loudon published in August 1999).

In response to Appellant argument, Oliver discloses as described previously in the final action mailed on April 21, 2006 that Oliver discloses the method and apparatus to perform memory management in an object-oriented programming (See the

Abstract). More specifically, for the limitation "comparing the value of the next pointer to the value of the memory location of the smart pointer in which the next pointer is included", as indicated by the examiner that the Oliver compares/examines next and previous pointer in the memory which could only be done by comparing the memory locations/addresses of the pointers (see the attached for the definition of address from Microsoft dictionary). In addition, Oliver discloses the pointers next and previous are examined (compared) before they are deleted from the memory (col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer" and "previous pointer" pointers are examined (compared)... If the "next pointer" pointer is the same as the "previous pointer" then there is clearly only one pointer remaining in the list... final pointer is deleted, no pointers will remain in the list. Thus, if the "next pointer" pointer is the same as the "previous pointer" pointer, then the object... deleted... and the last pointer to the object... deleted... If the "next pointer" pointer is not the same as the "previous pointer"... other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506") (Emphasis added). **Appellant further argued and in response to Appellant argument, Oliver discloses the limitation** "deleting the memory-resident element associated with the smart pointer (col. 5, lines 28-29 "the object (memory resident element) is then deemed unreferenced and may be deleted" and col. 5, lines 22-23 "delete a pointer... is deleted") if the value of the next pointer of the smart pointer is equal to the value of the memory location of the smart pointer in which the next pointer is included (col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer" and "previous pointer" pointers are examined (compared)... If the "next pointer" pointer is the same as the "previous pointer" then there is clearly only one pointer remaining in the list... final pointer is deleted) and not deleting the memory-resident element if the value of the next pointer of the smart pointer is not equal to the value of the memory location of the smart pointer in which the pointer is included (col. 5, lines 43-46 "If the "next pointer" pointer is not the same as the "previous pointer" pointer, however, then there are clearly other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506")". Therefore, the Examiner considers Oliver does disclose the limitations as claimed in claim 3.

Respecting Claims 13, 15-22 (page 16)

But, as the Examiner already acknowledged, the cited text of Oliver at col. 5, lines 34-37 compares the next and previous pointers. The "value of the memory location of the smart pointer" is not even considered or mentioned in the cited text of Oliver. It is not used in the Oliver test. Only the next and previous pointers are compared in Oliver. Hence, Oliver unquestionably fails to disclose at least Applicant's claimed feature of "comparing the value of the memory location of the smart pointer to the value of the next-pointer of the smart pointer." Likewise, Oliver fails disclose the step of "deleting the memory-resident element associated with the smart pointer if the value of the next-pointer of the smart pointer is equal to the value of the memory location of the smart pointer in which the next-pointer is included and not deleting the memory-resident element if the value of the next-pointer of the smart pointer is not equal to the value of the memory location of the smart pointer in which the next-pointer is included" as recited in claim 13. For at least these reasons, the rejection of claim 13 in view of Oliver is deficient, as Oliver fails to disclose each and every element recited in claim 13.

Examiner's Response

In response to Appellant argument, Oliver discloses as described previously in the final action mailed on April 21, 2006 that Oliver discloses the method and apparatus to perform memory management in an object-oriented programming (See the Abstract). More specifically, for the limitation "comparing the value of the next pointer to the value of the memory location of the smart pointer in which the next pointer is included", as indicated by the examiner that the Oliver compares/examines next and previous pointer in the memory which could only be done by comparing the memory locations/addresses of the pointers (see the attached for the definition of address from Microsoft dictionary). In addition, Oliver discloses the pointers next and previous are examined (compared) before they are deleted from the memory (col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer" and "previous pointer" pointers are examined (compared)... If the "next pointer" pointer is the same as the "previous pointer" then there is clearly only one pointer remaining in the list... final pointer is deleted, no pointers will remain in the list. Thus, if the "next pointer" pointer is the same as the "previous pointer" pointer, then the object... deleted... and the last pointer to the object... deleted... If the "next pointer" pointer is not the same as the "previous pointer"... other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506") (Emphasis added). **Appellant further argued and in response to Appellant argument, Oliver discloses the limitation "**deleting the memory-resident element associated with the smart pointer (col. 5, lines 28-29 "the object (memory resident element) is then deemed unreferenced and may be deleted" and col. 5, lines 22-23 "delete a pointer... is deleted") if the value of the next pointer of the smart pointer is equal to the value of the memory location of the smart pointer in which the next pointer is included (col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer" and "previous pointer" pointers are examined (compared)... If the "next pointer" pointer is the same as the "previous pointer" then there is clearly only one pointer remaining in the list... final pointer is deleted) and not deleting the memory-resident element if the value of the next pointer of the smart pointer is not equal to the value of the memory location of the smart pointer in which the pointer is included (col. 5, lines 43-46 "If the "next pointer" pointer is not the same as the "previous pointer" pointer, however, then there are clearly other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506")". Therefore, the Examiner considers Oliver does disclose the limitations as claimed in claim 13.

Respecting Claims 23, 25-28 and 30-33 (page 18-19)

But, as the Examiner already acknowledged, the cited text of Oliver at col. 5, lines 34-37 compares the next and previous pointers. The "value of the memory location of a selected smart pointer giving up its association" is not even considered or mentioned in the cited text of Oliver. It is not used in the Oliver test. Only the next and previous pointers are compared in Oliver. Hence, Oliver unquestionably fails to disclose at least Applicant's claimed feature of "comparing the value of the memory location of a selected smart pointer giving up its association with the memory-resident element to the value of the next-pointer of the selected smart pointer." Likewise, Oliver fails disclose the step of "providing a deletion means for deleting the memory-resident element associated with the smart pointer if the value of the next-pointer of the selected smart pointer is equal to the value of the memory location of the selected smart pointer in which the next-pointer is included and not deleting the memory-resident element if the value of the next- pointer of the selected smart pointer is not equal to the value of the memory location of the selected smart pointer in which the next pointer is included" as recited in claim 23. For at least these reason, the rejection of claim 23 in view of Oliver is deficient, as Oliver fails to disclose each and every element recited in claim 23.

Examiner's Response

In response to Appellant argument, Oliver discloses as described previously in the final action mailed on April 21, 2006 that Oliver discloses the method and apparatus to perform memory management in an object-oriented programming (See the Abstract). More specifically, for the limitation "comparing the value of the next pointer to the value of the memory location of the smart pointer in which the next pointer is included", as indicated by the examiner that the Oliver compares/examines next and previous pointer in the memory which could only be done by comparing the memory locations/addresses of the pointers (see the attached for the definition of address from Microsoft dictionary). In addition, Oliver discloses the pointers next and previous are examined (compared) before they are deleted from the memory (col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer" and "previous pointer" pointers are examined (compared)... If the "next pointer" pointer is the same as the "previous pointer" then there is clearly only one pointer remaining in the list... final pointer is deleted, no pointers will remain in the list. Thus, if the "next pointer" pointer is the same as the "previous pointer" pointer, then the object... deleted... and the last pointer to the object... deleted... If the "next pointer" pointer is not the same as the "previous pointer"... other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506")

(Emphasis added). **Appellant further argued and in response to Appellant argument, Oliver discloses the limitation** "deleting the memory-resident element associated with the smart pointer (col. 5, lines 28-29 "the object (memory resident element) is then deemed unreferenced and may be deleted" and col. 5, lines 22-23 "delete a pointer... is deleted") if the value of the next pointer of the smart pointer is equal to the value of the memory location of the smart pointer in which the next pointer is included (col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer" and "previous pointer" pointers are examined (compared)... If the "next pointer" pointer is the same as the "previous pointer" then there is clearly only one pointer remaining in the list... final pointer is deleted) and not deleting the memory-resident element if the value of the next pointer of the smart pointer is not equal to the value of the memory location of the smart pointer in which the pointer is included (col. 5, lines 43-46 "If the "next pointer" pointer is not the same as the "previous pointer" pointer, however, then there are clearly other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506")". Therefore, the Examiner considers Oliver does disclose the limitations as claimed in claim 23.

<u>Respecting Claim 35</u> (page 22)

But, as the Examiner already acknowledged, the cited text of Oliver at col. 5, lines 34-37 compares the next and previous pointers. The "the memory location of the smart pointer in which the selected previous pointer is included" is not even considered or mentioned in the cited text of Oliver. It is not used in the Oliver test. Only the next and previous pointers are compared in Oliver. Hence, Oliver unquestionably fails to disclose at least Applicant's claimed feature of "comparing the value of the previous pointer to the value of the memory location of the smart pointer in which the selected previous pointer is included." Likewise, Oliver fails disclose the step of "deleting the memory-resident element associated with the smart pointer if the value of the previous pointer of the smart pointer is equal to the value of the memory location of the smart pointer in which the previous pointer is included and not deleting the memory-resident element if the value of the previous pointer of the smart pointer is not equal to the value of the memory location of the smart pointer in which the previous pointer is included" as recited in claim 35. For at least these reason, the rejection of claim 35 in view of Oliver is deficient, as Oliver fails to disclose each and every element recited in claim 35.

<u>Examiner's Response</u>

In response to Appellant argument, Oliver discloses as described previously in the final action mailed on April 21, 2006 that Oliver discloses the method and apparatus to perform memory management in an object-oriented programming (See the Abstract). More specifically, for the limitation "comparing the value of the next pointer to the value of the memory location of the smart pointer in which the next pointer is included", as indicated by the examiner that the Oliver compares/examines next and previous pointer in the memory which could only be done by comparing the memory locations/addresses of the pointers (see the attached for the definition of address from Microsoft dictionary). In addition, Oliver discloses the pointers next and previous are examined (compared) before they are deleted from the memory (col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer" and "previous pointer" pointers are examined (compared)... If the "next pointer" pointer is the same as the "previous pointer" then there is clearly only one pointer remaining in the list... final pointer is deleted, no pointers will remain in the list. Thus, if the "next pointer" pointer is the same as the "previous pointer" pointer, then the object... deleted... and the last pointer to the object... deleted... If the "next pointer" pointer is not the same as the "previous pointer"... other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506") (Emphasis added). **Appellant further argued and in response to Appellant argument, Oliver discloses the limitation** "deleting the memory-resident element associated with the smart pointer (col. 5, lines 28-29 "the object (memory resident element) is then deemed unreferenced and may be deleted" and col. 5, lines 22-23 "delete a pointer... is deleted") if the value of the next pointer of the smart pointer is equal to the value of the memory location of the smart pointer in which the next pointer is included (col. 5, lines 34-37 "FIG. 5D... Prior to deleting a pointer, the "next pointer" and "previous pointer" pointers are examined (compared)... If the "next pointer" pointer is the same as the "previous pointer" then there is clearly only one pointer remaining in the list... final pointer is deleted) and not deleting the memory-resident element if the value of the next pointer of the smart pointer is not equal to the value of the memory location of the smart pointer in which the pointer is included (col. 5, lines 43-46 "If the "next pointer" pointer is not the same as the "previous pointer" pointer, however, then there are clearly other pointers remaining that point to the object. In this case, a pointer is removed from the list in step 508 and then the pointer is deleted in step 506")".

Therefore, the Examiner considers Oliver does disclose the limitations as claimed in claim 35.

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

Satish S. Rampuria

[illegible]

EDDIE C. LEE
SUPERVISORY PATENT EXAMINER

Conferees:

Eddie Lee (Appeal brief specialist)

Mary Steelman (Primary Examiner) Mary Steelman Primary Examiner

Satish S. Rampuria (The Examiner) [illegible]